Università degli Studi di Napoli Parthenope

Scuola Interdipartimentale delle Scienze, dell'Ingegneria e dellaSalute

Dipartimento di Scienze e Tecnologie

Corso di Laurea Triennale in Informatica

Torrent

Progetto di Reti dei Calcolatori

Candidati:

Pierluca Landolfi

Lorenzo Pelliccia

ANNO ACCADEMICO 2023/2024

Indice

Traccia Progetto

Progettare una rete P2P con protocollo TCP che simula lo scambio file utilizzata da torrent.
Ci sono n peers (n>2) che mettono a disposizione un file di testo (tutti i peers hanno lo stesso file) nel momento in cui un nuovo peers entra nella rete e richiede il file questo deve essere inviato da tutti i peers che hanno a disposizione quel file. Ogni peers invia una sottoporzione del file, per semplicità è necessario dividere la lunghezza del file per il numero di peers che hanno il file.

Es.

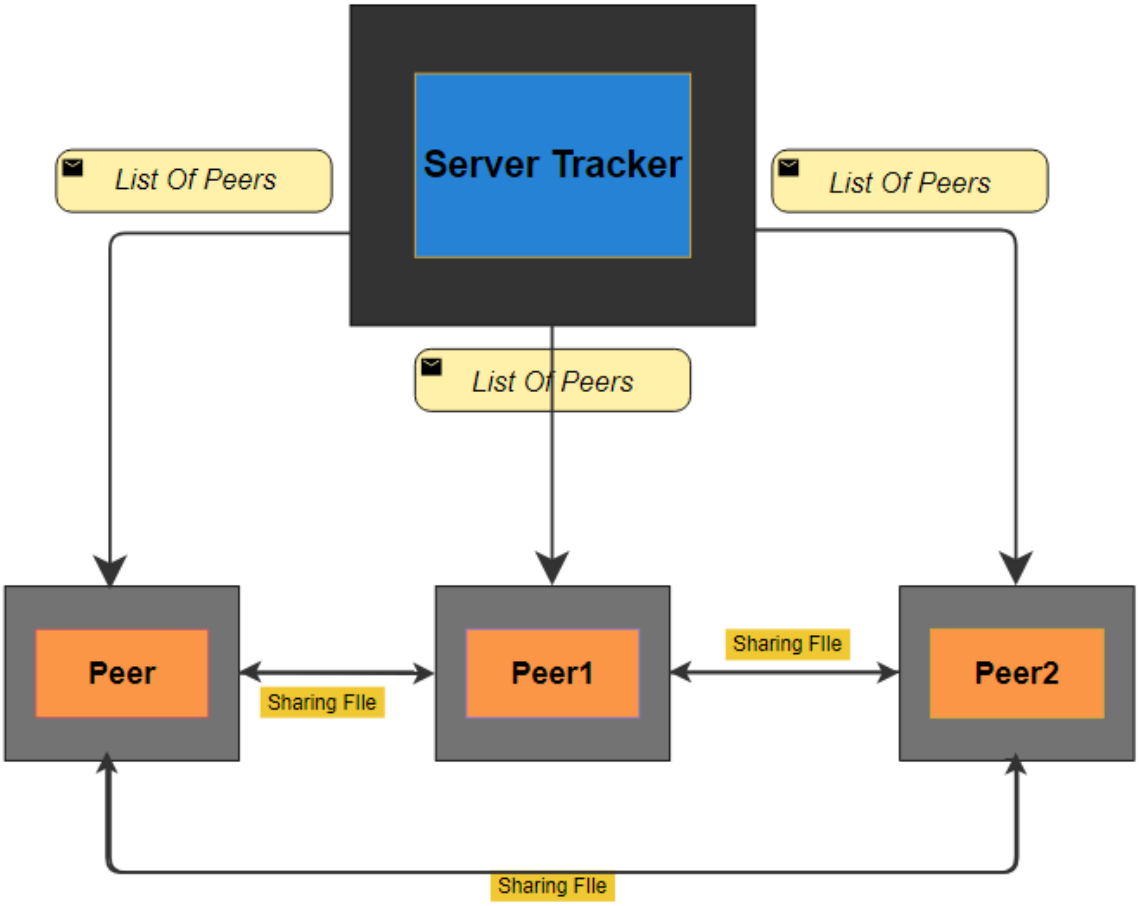
- 3 peers hanno il file
  - I peers avranno id: 1, 2, 3
- La dimensione del file viene divisa per tre arrotondando il numero di bytes da trasmettere
  - filesize 1024mb / 3 = 341,33
  - Peer 1 invia 341 bytes
  - Peer 2 invia 341 bytes
  - Peer 3 invia 342 byte

Aggiungere un server tracker che invia la lista dei peers che hanno il file.

Descrizione del progetto

Il progetto consiste nella realizzazione di una rete peer – to – peer basata sul protocollo TCP che simula il sistema di condivisione file utilizzato da torrent.
Ogni peers possiede una copia dello stesso file ed un ID sequenziale che lo contraddistingue dagli altri.
Il sistema permette ai peers di leggere e scaricare il file in Byte.
Il Server Tracker tiene traccia dei peer che hanno il file nel momento in cui un nuovo peers decide di collegarsi alla porta per ricevere il file.
La dimensione del file viene suddivisa per il numero di peers presenti all'interno del tracker in modo tale che ogni peers invia una sottoporzione del file al peers che lo richiede.

Descrizione e schema dell'architettura



In questo schema si evidenzia l'utilità del Server Tracker che è il cuore del sistema ed è visto come registro centrale per i peer nella rete. La sua responsabilità principale è mantenere e fornire l'elenco aggiornato dei peer attivi. I peer si connettono al Server Tracker per ottenere questa informazione che è essenziale per stabilire le connessioni peer – to – peer per la condivisione dei file.
Ogni peer nel sistema P2P è sia un client che un server. Come client, richiede al server Tracker l'elenco dei peer e poi utilizza queste informazioni per connettersi direttamente agli altri peer nella rete e scaricare parti di file. Come server, ogni peer accetta connessioni in entrata da altri peer e condivide con loro parti di file. Questo viene rappresentato dalla freccia bidirezionale tra i diversi peer.

Dettagli implementativi del client/server

La classe **Peers** va ad estendere Thread permettendo ad ogni peer di operare in maniera concorrente.
All'interno del costruttore di questa classe si crea una server socket per accettare connessioni da altri peers sulla porta specificata, inoltre, viene assegnato l'ID ai peers in maniera progressiva.
Viene inizializzato un array di tipo **Byte** che permette di contenere i byte di un file. Nel momento in cui risulta essere vuoto, se il primo peer entra, legge l'intero file dal disco essendo l'unico attualmente presente.
Nel momento in cui l'array di tipo Byte viene riempito, il server accetta una connessione in arrivo da un altro peer per poi andare a recuperare la lista dei peers che sono connessi al tracker per permettere la divisione del file in parti uguali.

I successivi peers scaricano parti del file dagli altri peer presenti nel tracker andando ad utilizzare il metodo **getListOfPeers** presente all'interno della classe per ottenere la lista dei peers presenti all'interno del Server Tracker.

Per calcolare la porzione del file, tenendo conto del numero di peers che in quel momento detengono il file, recupera la lista dei peers connessi al tracker e successivamente divide il file in sotto porzioni basandosi sulla dimensione totale del file e sul numero di peer da cui scaricarlo.

Viene effettuato un ciclo for per i peer presenti all'interno del tracker dove viene inizializzato un array temporaneo di tipo Byte per contenere la porzione di file dal peer corrente. Per fare questo viene richiamata la funzione **returnFile** che identifica il peer da cui scaricare la porzione di file e stabilisce una connessione con il peer specificato.

Viene inizializzata una flag contenente la parola chiave "**volatile**" di tipo boolean in modo tale da assicurare che ogni lettura di tale variabile da parte di un thread ottenga l'ultimo valore scritto da qualsiasi thread. Senza la parola chiave "**volatile**" potremmo avere una visione inconsistente dello stato del download tra i diversi thread.
La classe **Server Tracker** estende anch'essa Thread permettendogli di operare in modo concorrente e gestire le richieste di connessione da parte dei peers.

All'interno di questa classe viene definito un metodo per notificare l'aggiunta di un nuovo peer alla lista .

All'interno del costruttore di questa classe si istanzia una Server Socket che resta in ascolto sulla porta che viene specificata.

In conclusione, è possibile dire che un peer quando entra all'interno del sistema richiede la lista dei peer che hanno il file per poi andarlo a leggere e scaricare per poi entrare anch'esso nella lista dei peer che hanno il file in modo tale che, qual ora dovesse arrivare un ulteriore peer, faccia la medesima richiesta, con l'aggiunta del peer precedente che ora detiene anch'esso il file.

Parti rilevanti del codice sviluppato

Metodo returnFile

```
public byte[] returnFile(int port, int size) throws IOException {
    Socket socketStream = new Socket();

    InetSocketAddress peersAddress = new InetSocketAddress(port);

    socketStream.connect(peersAddress);

    InputStream stream = socketStream.getInputStream();

    byte[] buffer = new byte[size];

    stream.read(buffer);

    stream.close();
    socketStream.close();

    return buffer;
}
```

Questo metodo prende in input la porta del peer e la parte di file in questione.

1. Crea una nuova socket.
2. Viene creato un endpoint sulla porta che viene passata in input per identificare il peer da cui scaricare la porzione di file.
3. Si stabilisce una connessione TCP al peer specificato.
4. Viene inizializzato uno stream per ricevere dati dal peer.
5. Si utilizza un buffer di tipo byte per memorizzare i dati ricevuti dal peer.
6. Vengono letti i dati dal stream e vengono memorizzati nel buffer
7. Viene chiuso lo stream ed il socket per rilasciare le risorse di rete associate alla connessione
8. Si restituisce il buffer contenente i dati scaricati.

Calcolo dimensione

```
    sliceSize = fileSize / peers.size();

    if ((this.id - 1) * sliceSize + sliceSize > fileSize) {
        sliceSize = fileSize - (this.id - 1) * sliceSize;
    }
    byte[] temp = new byte[sliceSize];
    DataOutputStream outputStream = new DataOutputStream(connection.getOutputStream());

    for(int i = 0; i < sliceSize; i++) {
        temp[i] = fileToShare[(this.id - 1) * sliceSize + i];
    }
    outputStream.write(temp);
}
```

Permette di calcolare la dimensione della porzione di file da inviare al peer.

1. **sliceSize** permette di calcolare la dimensione di ciascuna porzione di file per assicurarsi che ogni peer riceva una parte equa del file.
2. Si controlla se il peer è l'ultimo della lista e potrebbe non dover inviare una porzione completa.
3. Si calcola la nuova dimensione sottraendo la quantità di byte già assegnata ai peer precedenti dalla dimensione totale del file
4. Si crea un array di byte temporaneo per memorizzare la porzione di file da inviare
5. Prepara un flusso di output per inviare dati al peer connesso.
6. Ciclo che permette di copiare il byte corrispondente dal file condiviso all'array temporaneo garantendo che ogni peer riceva la parte corretta del file.
7. Invio la porzione di file al peer connesso

Ricezione e Assemblaggio file

```java
int j = 0;
for (Integer peer : peers) {
    byte[] temp;
    temp = returnFile(peer, sliceSize);
    for(int i = 0; i < sliceSize; i++) {
        if(j < fileSize) {
            fileToShare[j] = temp[i];
            j++;
        }
```

1. Inizializza un array temporaneo di tipo byte per ospitare la porzione di file ricevuta dal peer corrente
2. L'array temp richiede e riceve la porzione di file dal peer corrente.
3. Ciclo su ogni byte della porzione ricevuta
4. Controllo dell'indice per verificare che sia entro i limiti della dimensione totale del file
5. In caso affermativo assegna il byte corrente dalla porzione ricevuta alla posizione corrente nell'array del file completo

Lista dei Peers

```java
public List<Integer> getListOfPeers(int trackerPort) throws IOException, ClassNotFoundException {

    InetSocketAddress serverAddress = new InetSocketAddress(trackerPort);

    Socket connectionTracker = new Socket();
    connectionTracker.connect(serverAddress);

    ObjectInputStream stream = new ObjectInputStream (connectionTracker.getInputStream());

    List<Integer> peers = (List<Integer>) stream.readObject();

    stream.close();
    connectionTracker.close();

    return peers;
}
```

Metodo per ottenere la lista dei peers dal server Tracker prendendo in input la porta del tracker.

1. Si stabilisce una connessione con il tracker specificando l'indirizzo del tracker a cui il peer vuole connettersi per ottenere la lista dei peers attivi all'interno del tracker.
2. Si crea un oggetto Socket e si stabilisce una connessione TCP tra il peer ed il tracker sulla porta specificata tramite la primitiva connect.
3. Si recupera l'input stream della socket per far si che all'interno dell'oggetto letto ci sia una lista di peers attivi inviata dal tracker.
4. Si effettua il cast poiché il metodo readObject() restituisce un tipo Object.
5. Restituisce la lista di peer attivi ricevuta dal tracker.

Manuale utente su compilazione ed esecuzione

È stato utilizzato IntelliJ come IDEE di sviluppo del progetto.
Il linguaggio di programmazione utilizzato per lo sviluppo è Java.

Le librerie:

1. java.io.ObjectOutputStream: Questa classe è utilizzata per inviare oggetti che contengono informazioni sullo stato del download, liste di peer da cui scaricare parti del file. La classe serializza gli oggetti in byte per l'invio.
2. java.net.ServerSocket: Fa parte del package java.net e rappresenta un socket lato server. Un oggetto di tipo serversocket ascolta su una porta specifica le richieste in arrivo da parte dei client. Quando un client cerca di connettersi, il server accetta la connessione creando un nuovo socket che gestisce le comunicazioni con il client.
3. java.net.Socket: fa parte del package java.net e rappresenta un socket lato client. Una volta stabilita la connessione il client ed il server possono scambiare dati attraverso i rispettivi oggetti InputStream e OutputStream.
4. java.io.ObjectInputStream: fa parte del package java.io permette di leggere oggetti serializzati ovvero convertiti in una sequenza di byte, da uno stream di input. Viene utilizzata dal lato del ricevente per leggere e deserializzare gli oggetti.
5. Java.io.DataOutputStream: fa parte del package java.io, viene utilizzata per inviare parti di un file attraverso la rete da un peer all'altro.

Sono state utilizzate per la comunicazione socket in Java.
Altre librerie come:

1. Java.nio.file.Path: fa parte del pacchetto java.nio.file. Viene utilizzata per specificare la posizione dei file da condividere o scaricare
2. Java.nio.file.Files fa parte del pacchetto java.nio.file. Viene utilizzata per leggere il contenuto di un file da condividere in un array di byte e dopo aver scaricato parti di un file da altri peer viene utilizzato il metodo per scrivere queste parti del file sull'array di byte

Queste librerie vengono utilizzate per gestire file, percorsi dei file,
La libreria InetSocketAddress viene utilizzata per specificare gli indirizzi di connessione per i socket, sia per il server che per il tracker.

Uso dell'applicazione

Dopo l'avvio i peer inizieranno automaticamente a comunicare con il server Tracker per ottenere l'elenco dei peer attivi e coordinare la condivisione dei file. Gli utenti possono monitorare l'output sul terminale per vedere lo stato delle operazioni di condivisione e di download.
Per inserire un nuovo peer, bisogna creare un oggetto di tipo Peers passandogli in input la sua porta e la porta del tracker alla quale deve collegarsi, successivamente bisogna dargli il comando di avvio tramite il metodo .start().
È necessario inserire un ciclo per permettere la progressione del codice finché il peer non scarica il file.
Infine, il peer viene aggiunto alla lista dei peer che hanno il file invocando il metodo .notifyPeer("Nome peer aggiunto").
Il Main inizializza ed avvia il ServerTracker che comincia ad ascoltare le richieste dei peer. Crea le istanze dei peer e le avvia che si connettono al ServerTracker per ottenere l'elenco dei peer attivi iniziando il processo di condivisione del file. Inoltre, si assicura che i peer completino il download del file prima di procedere con l'avvio di nuovi peer.

Progetto realizzato da:

Nome: Pierluca
Cognome: Landolfi
Matricola: 0124002709

Nome: Lorenzo
Cognome: Pelliccia
Matricola: 0124002578